C# and Java Keyword Comparison

Comparing the keywords in C# and Java gives insight into major differences in the languages, from an application developer's perspective. Language-neutral terminology will be used, if possible, for fairness.

Equivalents

The following table contains C# and Java keywords with different names that are so similar in functionality and meaning that they may be subjectively called "equivalent." Keywords that have the same name and similar or exact same meaning will not be discussed, due to the large size of that list. The Notes column quickly describes use and meaning, while the example columns give C# and Java code samples in an attempt to provide clarity.

It should be noted that some keywords are context sensitive. For example, the *new* keyword in C# has different meanings that depend on where it is applied. It is not used only as a prefix operator creating a new object on the heap, but is also used as a method modifier in some situations in C#. Also, some of the words listed are not truly keywords as they have not been reserved, or may actually be operators, for comparison. One non-reserved "keyword" in C# is *get*, as an example of the former. *extends* is a keyword in Java, where C# uses a ':' character instead, like C++, as an example of the latter.

C# Keyword	Java Keyword	Notes	C# Example	Java Example

Base	super	Prefix operator that references the closest base class when used inside of a class's method or property accessor. Used to call a super's constructor or other method.	public MyClass(string s)
			: **base**(s)
			
			{
			}
			public MyClass()
			: **base**()
			{
			}
				Public MyClass(String s)
				{
				super(s);
				}
				public MyClass()
				{
				super();
				}

Bool	boolean	Primitive type which can hold either true or false value but not both.	**bool** b = true;	**boolean** b = true;

Is	instanceof	Boolean binary operator that accepts an l-value of an expression and an r-value of the fully qualified name of a type. Returns true iff l-value is castable to r-value.	MyClass myClass = new MyClass();
			if
			(myClass **is**MyClass)
			{
			//executed
			}
				MyClass myClass = new MyClass();
				if
				(myClass**instanceof** MyClass)
				{
				//executed
				}

lock	synchronized	Defines a mutex-type statement that locks an expression	MyClass myClass = new MyClass();
			lock (myClass)
			{
				MyClass myClass = new MyClass();
				synchronized (myClass)
				{

		(usually an object) at the beginning of the statement block, and releases it at the end. (In Java, it is also used as an instance or static method modifier, which signals to the compiler that the instance or shared class mutex should be locked at function entrance and released at function exit, respectively.)	`//myClass is`
			`//locked`
			`}`
			`//myClass is`
			`//unlocked`
				`//myClass is`
				`//locked`
				`}`
				`//myClass is`
				`//unlocked`

namespace	package	Create scope to avoid name collisions, group like classes, and so on.	**`namespace`** `MySpace`
			`{`
			`}`
				`//package must be first keyword in class file`
				`package` `MySpace;`
				`public class MyClass`
				`{`
				`}`

readonly	const	Identifier modifier allowing only read access on an identifier variable after creation and initialization. An attempt to modify a variable afterwards will generate a compile-time error.	`//legal initialization`
			`readonly` `int constInt = 5;`
			`//illegal attempt to`
			`//side-effect variable`
			`constInt = 6;`
				`//legal initialization`
				`const` `int constInt = 5;`
				`//illegal attempt to`
				`//side-effect variable`
				`constInt = 6;`

sealed	final	Used as a class modifier, meaning that the class cannot be subclassed. In Java, a method can also be declared final, which means that a subclass cannot override the behavior.	`//legal definition`
			`public` **`sealed`**`class A`
			`{`
			`}`
			`//illegal attempt to`
			`//subclass - A is`
			`//sealed`
			`public class B: A`
			`{`
			`}`
				`//legal definition`
				`public` **`final`** `class A`
				`{`
				`}`
				`//illegal attempt to`
				`//subclass - A is`
				`//sealed`
				`public class B extends A`
				`{`
				`}`

using	import	Both used for including other	**`using`** `System;`	**`import`** `System;`

		libraries into a project.		

internal	private	Used as a class modifier to limit the class's use inside the current library. If another library imports this library and then attempts to create an instance or use this class, a compile-time error will occur.	`namespace Hidden`
			`{`
			`internal` `class A`
			`{`
			`}`
			`}`
			`//another library`
			`using Hidden;`
			`//attempt to`
			`illegally`
			`//use a Hidden`
			`class`
			`A a = new A();`
				`package Hidden;`
				`private` `class A`
				`{`
				`}`
				`//another library`
				`import Hidden;`
				`//attempt to illegally`
				`//use a Hidden class`
				`A a = new A();`

:	extends	Operator or modifier in a class definition that implies that this class is a subclass of a comma-delimited list of classes (and interfaces in C#) to the right. The meaning in C# is very similar to C++.	`//A is a subclass`
			`of`
			`//B`
			`public class A :B`
			`{`
			`}`
				`//A is a subclass of`
				`//B`
				`public class A`**`extends`** `B`
				`{`
				`}`

:	implements	Operator or modifier in a class definition that implies that this class implements a comma-delimited list of interfaces (and classes in C#) to the right. The meaning in C# is very similar to C++.	`//A implements I`
			`public class A :I`
			`{`
			`}`
				`//A implements I`
				`public class A`**`implements`** `I`
				`{`
				`}`

Supported in C# but Not in Java

The following table enumerates keywords in C# that seem to have no equivalent atomic support in Java. If possible or interesting, code will be written in Java that simulates the associated C# support to demonstrate the keyword's functionality in C# for Java developers. It should be noted that this list is very subjective, because it is highly unlikely that two people working independently would arrive at the same comparison list.

C# Keyword	Notes	C# Example	Java Equivalent

as	Binary "safe" cast operator that accepts expression as an l-value and the fully qualified class type as the r-value. Returns corresponding reference of r-value type if castable else null.	`Object o = new string();`
		
		`string s = o` **`as`** `string;`
		`if (null != s)`
		`{`
		`//executed`
		`Console.writeln(s);`
		`}`
			`Object o = new String();`
			`string s = null;`
			`if (o instanceof String)`
			`{`
			`s = (String) o;`
			`}`
			`if (null != s)`
			`{`
			`//executed`
			`System.Out.Writeln(s);`
			`}`

checked	Creates a statement with one block, or unary expression operator. Requires the developer to catch any arithmetic exceptions that occur during block or expression evaluation.	`using System;`
		`short x = 32767;`
		`short y = 32767;`
		`checked`
		`{`
		`try`
		`{`
		`short z = y + z;`
		`}`
		`catch (OverflowException e)`
		`{`
		`//executed`
		`}`
		`}`	

decimal	Defines a 128 bit number.	**`decimal`** `d = 1.5m;`	
delegate	Very similar to a C++ function pointer "on steroids." Because of its complex nature, it will be discussed in more detail below.	**`delegate`** `void MyFunction();`	

enum	Very similar to *enum* in C++. Allows a developer to create a zero-relative type with a zero-relative named list. It is too bad that Java chose to not allow enums.	**`enum`** `colors {red, green, blue};`	`public class Colors`
			`{`
			`public static const Red = 0;`
			`public static const Green = 1;`
			`public static const Blue = 2;`
			`private int m_color;`
			`public Colors(int color)`
			`{`
			`m_color = color;`
			`}`

	They are somewhat important.		public void SetColor(int color)
			{
			m_color = color;
			}
			public int GetColor()
			{
			return (m_color);
			}
			}

event	Allows a developer to create event handlers in C#. Discussed more below.	public **event** MyEventHandler Handler;	

explicit	Used as a modifier for user-defined class operators converting the parameter type to this type. Similar to C++'s constructor accepting parameter type. Conversions with the *explicit*keyword imply that a client must explicitly use a cast operator for it to be called. Server code that defines the operator should use*explicit* if the conversion may cause an Exception or information loss	public class MyType
		{
		public
		static **explicit**operator
		MyType(int i)
		{
		//write code
		///converting int to
		//MyType
		}
		}
			public class MyClass
			{
			public MyClass(int i)
			{
			//write code to convert
			//this holding i
			}
			}

extern	Used as a modifier in an empty method definition, with the implementation usually existing in an external dll file. Similar to C++.	[DllImport("User32.dll")] public static **extern** int MessageBox(int h, string m, string c, int type);	

fixed	Must be used in "unsafe" mode for manipulating pointers (pointers	int[] ia = {1,2,3};
		fixed (int* i = &ia)
		{
		}	

	are allowed in C# but should be used sparingly).		
foreach	Defines a looping statement in C# for collections implementing specific enumeration interfaces. Very nice language feature used when every element in an enumeration will be inspected. Any necessary casting is done implicitly for the developer in case of generic container use. Compare to an equivalent Java code segment, which requires the developer to explicitly cast during inspection.	`using System.Collections;` `ArrayList list = new ArrayList();` `list.Add(1);` `list.Add(2);` **`foreach`** `(int i in list)` `{` `int j = i;` `}`	`Vector v = new Vector();` `v.addElement (new Integer(1));` `v.addElement(new Integer(2));` `for (int i = 0; i < v.size(); i++)` `{` `int j = (Integer)v.elementAt(i).toInt();` `}`
get*	Not truly a keyword (not reserved). Can be used as an identifier, but avoid. If used as `get { }` then defines a class accessor function. Very nice from the client's perspective, because it appears as if he is directly accessing some data in the class when he is not. Nice for the class writer because he can perform other functionality before returning data.	`class MyClass` `{` `private int m_int;` `public int MyInt` `{` **`get`** `{` `return m_int;` `}` `}` `MyClass m = new MyClass();` `Int m = m.MyInt;`	`class MyClass` `{` `private int m_int;` `public int getInt()` `{` `return (m_int);` `}` `}` `MyClass m = new MyClass();` `Int m = m.getInt();`

implicit	Similar to the *explicit* keyword defined above, but implies that a developer does not have to use an explicit cast for conversion. Converts the class to the parameter type. Similar to C++'s conversion operator.	class MyType
		{
		public
		static **implicit**operator int
		(MyType m)
		{
		//code to convert this to
		//int
		}
		}
			class MyType
			{
			public int getInt()
			{
			//write code to
			//convert
			//this to an int
			}
			}

in	Keyword prefix operator used in a*foreach* loop, described above. Provides readability and a signal to the compiler that the container will be to its right.	See `foreach` example.	

new*	The *new* keyword has a context-sensitive meaning in C#. While it is used as an operator that returns a reference to a newly created object in both languages, it is also used in C# as a modifier to hide previously defined methods, properties, and indexers, for example, in a base class with the same signature or name. Please read the documentation for more information.	public class MyClassBase
		{
		public virtual void foo()
		{
		}
		}
		public class MyClass :
		MyClassBase
		{
		public **new** void foo()
		{
		//hides base version
		}
		}
			public class MyClassBase
			{
			public void foo()
			{
			}
			}
			public class MyClass extends
			MyClassBase
			{
			//must create actually
			//new method signature
			public void foo2()
			{
			}
			}

object	Based on the object data type, used for boxing. (Note: I am not completely aware of all of the	**object** o = 1;	

	nuances between using this keyword and object at the time during writing this document. Please read Microsoft's online documentation.)		

operator	Keyword used in a class method overloading a supported operator. Operator overloading is not supported in Java.	`public class Vector3D`
		`{`
		`public static Vector3D`**`operator`**` + (Vector3D v)`
		`{`
		`return (new Vector3D(x+v.x,y+v.y,z+v.z));`
		`}`
		`}`
			`public class Vector3D`
			`{`
			`public Vector3D add(Vector3d two)`
			`{`
			`//add implementation`
			`}`
			`}`

out	Method parameter and caller modifier that signals that the parameter may be modified before return. Should be used sparingly.	`public class MyClass`
		`{`
		`public int sort(int[] ia,out int)`
		`{`
		`//add implementation`
		`}`
		`}`
		`int[] ia = {1,7,6};`
		`int i;`
		`int s = MyClass.sort(ia,`**`out`**`i);`	

override	Method or property modifier in C# that implies that this method should be called instead of the super class's virtual method in case a more generic reference is held at run-time.	`public class A`
		`{`
		`public virtual int Test()`
		`{`
		`return 0;`
		`}`
		`}`
		`public class B : A`
		`{`
		`public `**`override`**` int Test()`
		`{`
		`return 1;`
		`}`
		`}`
		`A a = new B();`
		`int I = a.Test(); //1 is returned`
			`public class A`
			`{`
			`public int Test()`
			`{`
			`return 0;`
			`}`
			`}`
			`public class B extends A`
			`{`
			`public int Test()`
			`{`
			`return 1;`
			`}`
			`}`
			`A a = new B();`
			`int I = a.Test();`
			`//1 is returned. All methods`

			`//in Java are virtual`

params	Method formal parameter modifier that allows a client to pass as many parameters to the method as he wants. Nice language addition similar to the . . . in C++.	`public class MyClass`
		`{`
		`public static void`
		`Params(`**`params`**` int[] list)`
		`{`
		`//add implementation`
		`}`
		`}`
		`MyClass.Params(1,3,7);`
			`public class MyClass`
			`{`
			`public static void`
			`ParamSimulate(int[] ia)`
			`{`
			`}`
			`}`
			`int[] ia = {1,3,7};`
			`MyClass.ParamSimulate(ia);`

ref	Similar to *out* parameter above, except a *ref* parameter is more like an "in/out" param: it must be initialized before the call, where it is not required to initialize an "out" parameter before the method call.	See `out` example, but use **`ref`**.	

sbyte	A signed byte between -128 to 127	`//define an sbyte and assign`
		`sbyte`` mySbyte = 127;`	

set*	Please see *get* above. Also not a keyword, but treat it like it is. Allows a client to set data on a class instance.	`public class MyClass`
		`{`
		`private int m_int;`
		`public int MyInt`
		`{`
		`set`
		`{`
		`m_int = `**`value`**`;`
		`//see value below`
		`}`
		`}`
		`}`
		`MyClass m = new MyClass();`
		`m.MyInt = 3;`
			`public class MyClass`
			`{`
			`private int m_int;`
			`public void set(int i)`
			`{`
			`m_int = I;`
			`}`
			`}`
			`MyClass m = new MyClass();`
			`m.set(3);`

sizeof	Prefix operator similar to C++, accepting an expression as an r-value. Should be used sparingly, as it	`int i = 3;`
		`int s = `**`sizeof`**`(i);`	

	is only supported in unsafe mode.		

stacalloc	Used for allocating a block of memory on the stack. Should be used sparingly, as it is only supported in unsafe mode.	`public static unsafe void Main()`
		`{`
		`int* fib = stackalloc int[100];`
		`int* p = fib;`
		`*p++ = *p++ = 1;`
		`for (int i=2; i<100; ++i, ++p)`
		`*p = p[-1] + p[-2];`
		`for (int i=0; i<10; ++i)`
		`Console.WriteLine (fib[i]);`
		`}`	

string	Alias for the System.String class.	**`string`** `s = new String();`	`String s = new String();`

struct	Similar to a struct in C++. Lightweight, where a constructor is only called if *new* is used to create.	**`struct`** `MyStruct`
		`{`
		`int MyInt;`
		`}`
			`class MyStructSimulate`
			`{`
			`private int m_int;`
			`public int get()`
			`{`
			`return (m_int);`
			`}`
			`}`

this*	Context sensitive. C# allows for indexers, while Java does not. But*this* in both also returns a reference to the actual class member.	`class MyClass`
		`{`
		`private int[] m_array;`
		`public int` **`this`**`[int index]`
		`{`
		`get`
		`{`
		`return (m_array[index]);`
		`}`
		`set`
		`{`
		`m_array[index] = value;`
		`}`
		`}`
		`}`	

typeof	Prefix operator accepting an expression as an r-value returning the runtime type of the object.	`MyClass m = new MyClass();`
		`Type t =` **`typeof`**`(m);`
		`//same as m.GetType();`
			`MyClass m = new MyClass();`
			`Class c = m.getClass();`

uint	Unsigned integer.	**uint** i = 25;	
ulong	Unsigned long.	**ulong** l = 125;	

unchecked	Opposite of "checked" above. No arithmetic exceptions must be caught in the block. This is the default behavior.	**Unchecked**
		{
		}	

unsafe	Defines an "unsafe" block of code in C#. Should be used sparingly.	public static **unsafe**unsafeMethod();	
ushort	Unsigned short.	**ushort** u = 7;	

using*	Context sensitive in C#. Defines a block on an expression, where it is guaranteed that dispose is called on the expression after the block is at the end.	MyClass m = new MyClass();
		using (m)
		{
		}	

value*	Proxy for a passed value into a set function. Please see *set* and *get*above .	public class MyClass
		{
		private int m_int;
		public int MyInt
		{
		set
		{
		m_int = **value**;
		}
		}
		}	

virtual	Method or property modifier in C#. Similar to C++. All Java methods are virtual by default, so Java does not use this keyword.	public class MyClassBase
		{
		public **virtual** int GetInt()
		{
		return 3;
		}
		}
			public class MyClassBase
			{
			//all methods virtual by
			//default.
			public int GetInt()
			{
			return 3;
			}
			}

Supported in Java but Not in C#

The following keywords are supported in Java but not in C#.

Java Keyword	Notes	Java Example	C# Equivalent
native	Since Java was designed to run on any supported operating system, this keyword allows for interoperability and importing code compiled in some other language.		
transient	Supposedly currently unused in Java.		

synchronized*	Context-sensitive keyword. When used as an instance method modifier, guarantees that the single instance mutex will be gained at function entrance and released just before the function exits. If used as a static method modifier, then the class mutex will be used instead. Also allowed as a class modified, which means that all class access is synchronized implicitly.	`public` **`synchronized`** `void LockAndRelease()`
		
		`{`
		`//instance lock`
		`//implicitly`
		`//acquired`
		`//write code here`
		`//instance lock`
		`//implicitly`
		`//released`
		`}`
			`public void LockAndRelease()`
			`{`
			`//lock must`
			`//be`
			`//called`
			`//explicitly`
			`lock(this)`
			`{`
			`//code`
			`//here`
			`}`
			`}`

throws*	Slightly different	`public void`	`public void`

	meaning in C# and Java. The exception must be caught by the client in Java if it is not a RuntimeException.	`foo() `**`throws`**`MethodNotFoundException`
		`{`
		`}`
			`foo()`
			`{`
			`}`